Bootstrapping a Trustworthy and Seamless Digital Engineering System

Ph.D. Preliminary Exam

James S. Wheaton

Advised by: Dr. Daniel Herber

Department of Systems Engineering
Walter Scott, Jr. College of Engineering
Colorado State University

April 17, 2023

Committee Members:
Dr. Erika Miller
Dr. Steve Simske
Dr. Vinayak Prabhu
Outline

1. Introduction
2. Operational Concept
3. Theory Development
4. HACK Design
5. Summary & Research Plan
James S. Wheaton

- B.S. Mechanical Engineering, Purdue University (2011)
- Former software engineer and consultant in ecommerce, big data, and blockchain
- Started Systems Engineering Ph.D. @ CSU in 2017, part-time remote
- Completing coursework Spring 2023 in the 72-credit-hour Ph.D. degree program
- Computer hobbyist since age 5
- Likes to study programming languages of all kinds
- Builds all software from source with hardened toolchains, wherever possible
Introduction
Digital Engineering Is an Integration Challenge

Digital Engineering currently relies on *interoperability* as the primary mechanism for constructing the Authoritative Source of Truth, e.g. with APIs and format interchange standards\(^1\).

---

**Figure**: Depiction of NASA JPL OpenCAE Environment (Adapted from Delp 2019)

---

\(^1\) Bajaj, Friedenthal, and Seidewitz 2022
We Build Our Computer Systems Like Cities

Figure: xkcd: Dependency (Munroe 2020)

Figure: The Error Avalanche (Adapted from Claxton, Cavoli, and C. Johnson 2005)
Cybersecurity is a “Mess” or Wicked Problem

US CISA Director has recently highlighted a “normalization of deviance” in the computing industry and called on vendors to provide systems that are secure-by-default and secure-by-design.\(^1\) The situation is untenable:

- Pervasive use of memory-unsafe languages, including in OS, compilers, security-critical components and theorem provers\(^2\)
- Common Vulnerabilities and Exposures are on the rise
- CPU microarchitecture side-channel vulnerabilities are unpatchable\(^3\)
- Internet architecture vulnerabilities & protocol ossification\(^4\)
- Trusting Trust attack remains ignored after 50 years\(^5\)
- Cyber-infrastructure is inherently insecure\(^6\)

---

Digital Engineering Has a Reverse Salient

The reverse salient is a set of critical problems\(^1\) whereby system components “fail to deliver the necessary level of technological performance thereby inhibiting the performance delivery of the system as a whole”:\(^2\)

- WIMP applications paradigm — essential functions are outsourced
- false dichotomy of user / developer
- inscrutable binary executable vs. sprawling source code
- physical centralization + lack of isolation, e.g. CPU
- sequential-first processing, e.g. CPU
- lack of integrated program documentation, test, and verification facilities
- plethora of ill-defined languages/formats
- security-by-obscurity

\(^1\) Hughes 1993 \(^2\) Dedehayir and Mäkineif 2008
Looking for the Escape Hatch

The systemic problem of trustworthy cyber-systems has been known for 25 years.\(^1\) Recent research efforts have attacked the mess from different perspectives:

- Fully Countering Trusting Trust through Diverse Double-Compiling\(^2\)
- DARPA Cyber-Assured Systems Engineering (CASE)\(^3\)
- DARPA Clean-slate design of Resilient, Adapative, Secure Hosts (CRASH)\(^4\)
- DARPA META-II\(^5\)
- DARPA Circuit Realization At Faster Timescales (CRAFT)\(^6\)
- Deep Specification\(^7\)
- Formally-verified stack from assembly language to CPU\(^8\)

---

1. McLean 1997; Council et al. 1999; Mundie et al. 2002; Spafford 2004
2. Wheeler 2009
3. Cofer 2021
4. *Clean-slate design of Resilient, Adapative, Secure Hosts (CRASH)* 2010; Chiricescu et al. 2013
5. *META-II* 2010
6. *Circuit Realization At Faster Timescales (CRAFT)* 2015
7. Appel et al. 2017
Research Questions

**RQ1** What are the gaps, barriers and cost drivers of engineering provably-correct cyber-systems?

**RQ2** Can these gaps be adequately addressed with today’s computing ecosystem?

**RQ3** What would a clean-slate digital engineering system that addresses the gaps and barriers look like?

**RQ4** Can we prove that such an architecture is seamless and trustworthy?
Operational Concept of Clean-slate DE System
DE Meta-Model

Digital Twins

Model
- Model-Based Production Planning
- Model-Based Definition
- Model-Based Systems Engineering

Virtual
- Virtual Production System
- Virtual Qualification
- Virtual Certification
- Virtual Operations
- Virtual Ecosystem

Solutions
- Virtual ecosystem

Needs
- As-Needed
- As-Offered
- As-Specified
- As-Designed
- As-Planned

Design
- Business Model
- Market (Mission) Model

Physical
- As-Built
- As-Certified
- As-Tested
- As-Delivered
- As-Supported

Physical Systems

Figure: The Boeing MBSE Diamond: Continuity of the system’s ‘Digital Thread’ (Adapted from Seal 2018)
DE Essential Functionality

DE practitioners need a predictable set of affordances for doing their work:

- **Mathematics**: matrices, equation solving, calculus, optimization, probability and statistics, discrete math, theorem proving
- **Science**: physical constants & models, simulations, experimental design, properties of matter
- **Engineering**: 3D geometry, finite-element analysis, fluid dynamics, thermodynamics, materials, reliability, systems modeling, units
- **Knowledge Engineering**: ontologies, authoritative data, rich media, process meta-models, query capabilities
- **Project & Program Management**: PERT, critical path method, EVM, Gantt charts, project economics and accounting
Human-Computer Interaction

We need to re-think HCI for human factors:

- WIMP breaks down at scale
- Applications enforce costly context switches, data incommensurability
- Everything-is-an-object with Capabilities is a simpler formalism
- Coherence of textual & graphical representations aids efficient, diverse uses
- Localization and accessibility must be designed-in from the beginning
- AI augmentation is an option, powerful in some contexts
Quality Attributes

- **seamless**: consistent and coherent interfaces throughout
- **trustworthy**: provenance of components is known, auditable and traceable; components reliably implement their specifications and carry proof certificates
- **elegant**: “a system that is robust in application, fully meeting specified and adumbrated intent, is well-structured, and is graceful in operation”
  - efficacy
  - efficiency
  - robustness
  - minimizing unintended consequences
- **convivial**: serve the operator and their community for creative and autonomous use, with the power to develop mastery

---

1 M. D. Watson, Mesmer, and P. Farrington 2019; M. Watson, Mesmer, and P. Farrington 2020; Madni 2012; M. D. Watson, Griffin, et al. 2014; M. D. Watson 2017

2 Voinea 2018
Theory Development
Understanding Tool Integration

**Figure:** Tool Integration Entity-Relationship Diagram (Adapted from Thomas and Nejmeh 1992)
LISP the Meta-Language

Language-oriented programming has “advantages for domain analysis, rapid prototyping, maintenance, portability, user-enhanceable systems, reuse of development work, while also providing high development productivity” \(^1\)

One of the guidelines of language-oriented programming is that it “enables creators of languages to enforce its variants. ...When a program consists of pieces of different languages, values flow from one context into another and need protection from operations that might violate their integrity.” \(^2\)

*Programming paradigms depending on need*: imperative, functional/declarative, symbolic, constraint/logic, array and stack, dataflow, query, metaprogramming. Gradual typing supports different phases of the system development lifecycle.

---

\(^1\) Ward 1994 \(^2\) Felleisen et al. 2018
Defining Seamless Architecture

- Are Interfaces everywhere fully defined and satisfied at every connection endpoint (Port)?
- Do Parts refine their imported types?
- Do Part Specifications prove out Ports are derivations of internal Parts and in Ports and Item Flows?
- Disparate interfaces are not exposed to the operator ("islands of functionality")

Figure: Interfaces in SysML v2 demonstrating seamlessness (Adapted from Friedenthal 2023)
Defining Trustworthy

Trustworthiness is a Quality Attribute related to reliability and security, and based on a set of measurable factors:

- Behavior is well-defined
- Side-channels are explicitly guarded where feasible
- Object Capabilities are ubiquitous for fine-grained security\(^1\)
- Components carry proof certificates, with traceability
- System must be independently verifiable against their specifications
- Bootstrappable, defended against Trusting Trust attacks

\(^1\) Rees 1995; Richardson, Carey, and Schuh 1993
Sketching the Bootstrap Process

Figure: Simplified view of HACK bootstrap process
Formal Proof Strategy

**Goal:** A trustworthy system is constructible from untrustworthy components.\(^1\)

- Untrustworthy components are diverse
- Untrustworthy components produce the same output for a given input
- Trustworthy components carry proof certificates
- Trustworthy components are auditable
- Untrustworthy components are replaceable by trustworthy components
- Trustworthy system has an independently-verifiable root-of-trust

Build from Wheeler 2009’s Diverse Double-Compiling formal proof to include more of the system components.

\(^1\) Rajendran, Sinanoglu, and Karri 2016; Cui et al. 2022; Sethumadhavan et al. 2015
Design of High-Assurance Computing Kit
Figure: HACK Specification Tree
## HACK Language Stack

- **Kernel LISP**
  - Interaction Nets
  - SAT/SMT solver (in hardware)
  - tiny hex seed
- **Propagation Nets**
- **Gradually-Typed Vau Calculi**
- **UTT theorem prover**
- **Milawa FOL theorem prover**
- **(Timed) Abstract State Machines**
- **HALISP meta-language**
  - quadstore
  - SysML v2
  - HALISP standard library
  - HALISP standard library
  - egglog
  - miniKanren
  - microKanren
- **engineering eDSLs**
  - standard knowledgebase
- **special-purpose hardware**
- **FPGAs**
- **Egglog**
HALISP Language Design

(def:function hello-world  ;; name of the pure function
  :doc      (document "A complete DocBook object or AsciiDoc string")
  :type     [ String :-> String ]
  :params   [ name ]
  :requires [ (> (length name) 0) ]
  :ensures  (= (length out)  ;; Hoare triple post-condition
                (+ (length "Hello, ")
                   (length name)
                   (length "!")
                ))
  :satisfies [ :FR/001 ]  ;; SysML Block "satisfies" Relationship
  :tests    [(test trivial-example
                :doc "Test that the name is inserted into the greeting."
                (= "Hello, World!"
                   (hello-world "World")))
             (ref:test :T/HW-002)]
  :version  { :major 0 :minor 1 :patch 0 })  ;; enforceable semantic versioning

;; Separation of specification and implementation
(def:function-body hello-world
  (str "Hello, " name "!")  ;; function body usually starts on a newline

Figure: HALISP integrates formal verification, systems eng. & project mgmt.
HACK HCI (1)

Interactive Session:

```
(+ 2 2 2 2 2)
```

```
10 (Integer)
```

Group Chat:

**Bob B.** What is the prelude module? never heard of it.

**James W.** The prelude module has some basic functions like first, second, third

**James W.** They are defined in HALISP instead of being primitives

**Bob B.** OK - what's a primitive?

**James W.** It's a function implemented in the virtual machine, written in the
host language.

**Bob B.** I need to read more of the documentation. hold on

Figure: Team collaboration with text/voice chat and screenshare is built-in
Where do I start modeling the system architecture?: Document

Need Understanding
Designing a System that satisfies needs requires that the Systems Engineers first elicit and define Stakeholder Needs. This process is not always that straightforward, and part of the importance is being explicit and formal in the definitions of Stakeholder Needs and Goals. These clear statements are stored in the model.

My Annotations:
- [120-223] A capabilities database stores the requirements and reference models
- [236-250] A requirement is a necessary capability of the system
- [360-418] For more details check the SEBoK wiki

Figure 1. ARCADIA system architecture illustrative model³

Figure: Knowledgebase is browseable, annotatable, with transclusion & object graph views
HACK HCI (3)

**HACK Requirements Specification Tree**

```
(require 'macro-cad/diagrams/tree)

(tree-diagram { :auto-layout :vertical-tree
   :title "HACK Requirements Specification Tree"
   :version { :major 0 :minor 1 :patch 0 } }
(node { :rid :macro-cad-system-requirements
   :root true
   :background-color "#000000"
   :text-color "#ffffff"
   :inputs ( [ (node { :rid :stakeholder-needs
      :background-color "#ececec" })
      (node { :rid :engineering-standards
      :background-color "#ececec" }) ] }
(node { :rid :knowledgebase-requirements }
   (node { :rid :knowledgebase/database-requirements })
   (node { :rid :knowledgebase/memex-requirements })
   (node { :rid :knowledgebase/metalibrary-requirements }))
(node { :rid :systems-modeling-metalanguage-requirements }
   (node { :rid :metalanguage/HALISP-requirements })
   (node { :rid :metalanguage/MDE-library-requirements })
   (node { :rid :metalanguage/math-library-requirements }))
(node { :rid :macro-cad-machine-requirements }
   (node { :rid :machine/VM-requirements })
   (node { :rid :machine/hardware-requirements })
   (node { :rid :machine/UI-requirements }))
```

**Interactive Session:**

```
{ (ref :macro-cad-system-requirements)
   SysML Package(showing metadata)
   { :type SysML/Package
     :name "HACK System Requirements"
     :description "A Package of categorized Packages of Requirements"
     :version { :major 0 :minor 1 :patch 1 }
     :date-created (date 2022 5 1)
   } (help :metrics)
}
```

**Actions Available:**

- Run Simulation
- Run Impact Assessment
- Show All Children
- Show Direct Relationships
- Show Uses in SysML Diagrams
- Verify System Properties
- Modify Object Metadata
- Configure Diagram
- Configure Auto-Layout
- View History
- Edit Tabular Data

**Activities:**

- Add Block/Part
- Add Requirement
- Add Package
- Add Relationship
- Add Comment
- Annotate
- Control Versions
- Find in Model Tree
- Validate
- Ask AI Advisor
- Publish

**Figure:** Textual and graphical representations are coherent; interactive programming session; contextual actions are listed
The HACK prototype is being developed with the capabilities to model the HACK system. Current functionality is testing the LISP meta-language approach:

- Small LISP implementation with REPL in Ada 2012
- Programmable graphics with SVG for diagrams and presentation slides
- Use of DocBook standard to define systems engineering document templates
- Website generation from given templates and source files
- SysML and project definitions
- Knowledgebase entry definitions
Summary & Research Plan
A Grand Challenge in Systems Engineering

- Digital Engineering requires a **Transdisciplinary Systems Engineering** approach!\(^1\)
- We need the right framing and goal to build support
- End-to-end formal verification enables certification of system and components’ correctness so they can be deemed *finished* or safely refactored

\[^1\] Mesmer et al. 2022

**Figure**: The FM9001 gate-level model corresponds to the high-level functional model (Moore 2003; Moore 2007)
Research Contributions

1. Operational concept of a trustworthy & seamless digital engineering system
2. Formal definitions of ‘trustworthy’ and ‘seamless’ within this context
3. Formal proof of soundness of the approach of bootstrapping a trustworthy system from untrustworthy components
4. An open-source SysML v2 model of the proposed system architecture with prototype for demonstrating key functionality
Papers In-Progress

1. “Seamless Digital Engineering: Motivating a Grand Challenge”
   - 80% complete
   - **Target conference**: INCOSE Western States Regional Conference 2023

2. “Digital Engineering Modeling Languages as LISP-Embedded Domain-Specific Languages”
   - 20% complete
   - **Target conference**: INCOSE Western States Regional Conference 2023

   - 50% complete
   - **Target conference**: INCOSE IS 2024
Timeline to Completion

**Summer 2023**  JPL internship at half-time; continue HACK architecture modeling and prototype development; theory development, setup and testing of formal definitions and proofs; notification of acceptance for 2 conference papers

**Fall 2023**  Develop proofs, test and interpret results; architecture model completeness assessment; attend conference, and submit 1-2 papers to INCOSE IS conference or SE journal

**Spring 2024**  Final editing and presentation of results in dissertation; defense and graduation planned near end of semester
Future Work

• Continue architecture design and prototyping
• Reference model and reference architecture of DE lifecycle with product-line architecture instantiations for particular types of systems
• Computational type theory view and investigations of system architecture building from SysML v2 semantics
• Proving an architecture seamless and trustworthy at the type level
• Analysis of system bootstrap paths, size and effort
• Comparison and type-level analysis of extant computer system architectures
• Working out details of computational design of HACK systems
• Characterize ‘architectural ossification’ or intransigence
Thank you!

Figure: Places to intervene in a system (Adapted from Meadows 2008)
References


M. Bajaj, S. Friedenthal, and E. Seidewitz (2022). “Systems modeling language (SysML v2) support for digital engineering”. Insight 25.1. DOI: 10.1002/inst.12367

References (continued)


References (continued)


References (continued)


References (continued)

- M. Lipp et al. (2020). “Meltdown: Reading kernel memory from user space”. *Communications of the ACM* 63.6

F. Massacci, T. Jaeger, and S. Peisert (2021). “Solarwinds and the challenges of patching: Can we ever stop dancing with the devil?” *IEEE security & privacy* 19.2. DOI: 10.1109/MSEC.2021.3050433


References (continued)


- C. Mundie et al. (2002). *Trustworthy computing*. Tech. rep. Microsoft


References (continued)


References (continued)


References (continued)


References (continued)


References (continued)


